

Assembly & A86 Assembler



CMPE 230 - Spring 2023

Gökçe Uludoğan

Based on the slides by Abdullatif Köksal, with his permission.

Assembly

- Assembly language is *any* **low-level programming language** in which there is a very strong correspondence between the instructions in the language and the architecture's machine code instructions.

What is **machine code**?

Machine code is a computer program written in machine language instructions that can be executed directly by a computer's central processing unit (CPU).

Assembly

- Assembly language is *any* **low-level programming language** in which there is a very strong correspondence between the instructions in the language and the [architecture's machine code instructions](#).
- Because assembly depends on the machine code instructions, **every assembler has its own assembly language which is designed for exactly one specific computer architecture**.
- Assembly code is converted into executable machine code by a [utility program](#) referred to as an [assembler](#).

A86 Assembler

- It is designed for the Intel 86-family of microprocessors.
- You have to install XP as a virtual machine.

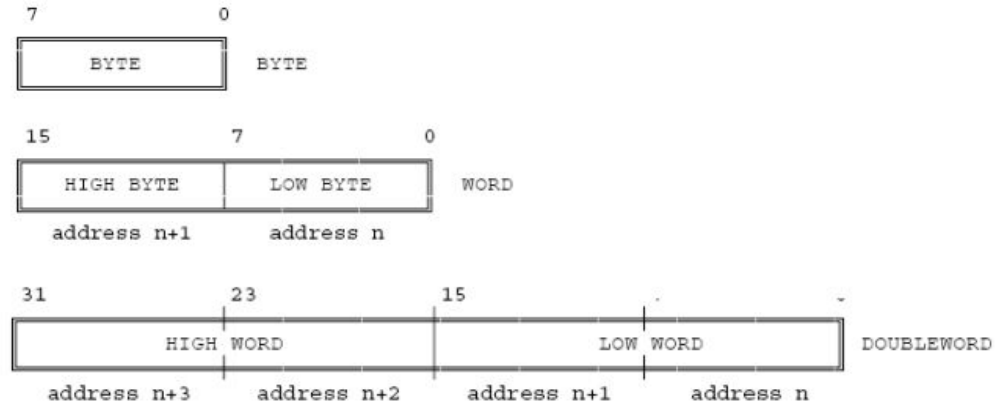
<https://zenodo.org/record/7654639>

Alternative link:

https://drive.google.com/file/d/1K1egWQzz8vB_ow41hftyBeEGt80YJw6j/view?usp=share_link

Bit, Byte, Words, etc

- Bit: Represent two values (0, 1)
- Byte: 8 bits
- Word: 2 bytes, 16 bits
- Double word: 4 bytes, 32 bits



Variables

- A variable is a unit of program data residing at a specific location in memory.
- You can define variables in A86 like in Java.
- You can find address of a variable with OFFSET command.

```
READ_FLAG      DB      0          ; Byte variable, initial value 0
FILE_COUNT     DW      ?          ; Word variable, no initial value
POPULATION     DD      100000     ; Dword variable, initial value 100000
MSG            DB      "Hello"    ; We can define string variable with a value
MSG            DB      "Hello World.$"
```

Registers

- We have 4 general purpose 2 byte (1 word) registers
- Each of them can be divided into two (high & low)

AX = AH AL

BX = BH BL

CX = CH CL

DX = DH DL

- We also have segment, pointer, and data transfer registers. For details, check out the references.

Variables vs Registers

- You can use variables to store data. However, arithmetic operations have to execute on registers.
- You can move data between variables and registers but you have to **check out the data type**.

MOV: Move

- It copies a piece of data from one location to another.

Mnemonic	Operands	Size	Operation	Flags
MOV	DEST, SRC	B/W	(DEST) <- (SRC)	None

MOV: Move

- It copies a piece of data from one location to another.

AX =	AH	AL
BX =	BH	BL
CX =	CH	CL
DX =	DH	DL

```
MOV BX,AX    ; Copies the contents of the AX register into the BX register.
MOV CH,DH    ; Copies the top byte of the DX register into the top byte of the CX register.
MOV BH,DL    ; Copies the bottom byte of the DX register into the top byte of BX.
MOV AH,12    ; Puts the value 12 decimal into the top half of the AX register.
MOV AH,0Ch   ; Does the same as the above except that the number is given in hexadecimal.
              ; Hexadecimal numbers MUST begin with a digit and end with a "h".
```

MOV: Move

- It copies a piece of data from one location to another.
- It can copy characters too (directly or via their ASCII Table).

```
MOV DL,"*" ; This puts the character "*" into DL (lower half of DX).
```

```
MOV DL,42 ; This does the same thing, as characters are stored as numbers. ( ASCII char 42 =  
"*)
```

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(72	48	110	H	104	68	150	h
9	9	11		41	29	51)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

MOV: Move

- It copies a piece of data from one location to another.
- It can copy characters too (directly or via their ASCII Table).
- It can copy address of a variable. (*OFFSET*)

```
MSG    DB    "Hello World.$"           ; A string variable with a value.  
  
MOV DX, OFFSET MSG                     ; DX points to (holds the address of) the string
```

MOV: Move

- It copies a piece of data from one location to another.
- It can copy characters too(directly or via their ASCII Table).
- You have to be careful about data size.

```
MOV AX,BH      ; Invalid operation, as you cannot move an 8 bit quantity to a 16 bit one.
MOV CH,BX      ; Similar to above, you cannot put a 16 bit quantity into an 8 bit one.
MOV 12,DL      ; You cannot put a value into the number 12. If you see this is a program what
               ; is probably meant is MOV DL,12 - put 12 into DL.
MOV DL,AL,CL   ; You cannot have 3 operands!
MOV AH        ; Neither can you have only 1! You must have exactly 2: destination and source
```

Stack: PUSH, POP

- A temporary storage area in memory to quickly store and retrieve data
- Pushing a number in a stack, and popping in the reverse order.

```
PUSH AX ; AX onto stack
```

```
PUSH DX ; DX onto stack
```

```
POP BX ; pop into BX
```

```
POP CX ; pop into CX
```

INT: Interrupt

- It calls an interrupt (like a function) to perform a special task.
 - Take input
 - Print to screen
 - Exit
- You have to put some values in registers to indicate the specific task.

INT: Interrupt - Useful Subfunctions

Interrupt.	SubFunction.	Input.	Output.
10h (VIDEO_INTERRUPT)	00 (SET_MODE) Sets the Video mode	AL=mode number -	-
	0Ch (WRITE_DOT) Puts a dot on the screen Graphics modes only	DX=row CX=column AL=colour	-
	0Dh (READ_DOT) Reads a dot on screen Graphics modes only	DX=row CX=column -	AL=colour - -
16h (KBD_IO)	00 (AWAIT_CHAR) Reads a character from keyboard	-	AL=character AH=scan_code
	01 (PREVIEW_KEY) Checks to see if a key is ready Does not remove key from buffer	-	Zero flag set - key ready AL=character AH=scancode
21h (DOS_INTERRUPT)	01 (KEYBOARD_INPUT) Reads and displays one character	-	AL=character read -
	02 (DISPLAY_OUTPUT) Displays one character on screen	DL=character -	-
	08 (NO_ECHO_INPUT) Same as 01 but not displayed	-	AL=character -
	09 (PRINT_STRING) Displays a string on screen String must end with "\$"	DX=address of string - -	-
	0A (BUFFERED_INPUT) Reads a string from keyboard	DX=address of buffer First character=max length -	Second char of buffer=length of input Rest of buffer=input string followed by carriage return (0Dh)
	4Ch (EXIT)	AL=exit code	-

INT: Interrupt - DOS Interrupts (21h)

Subfunction (AH)	Input	Output
01 (KEYBOARD_INPUT) Reads and displays one character	-	AL=character read
02 (DISPLAY_OUPUT) Displays one character on screen	DL=character	-
08 (NO_ECHO_INPUT) Same as 01 but not displayed	-	AL=character read
09 (PRINT_STRING) Displays a string on screen String must end with "\$"	DX=address of string	-
0A (BUFFERED_INPUT) Reads a string from keyword	DX=address of buffer First character=max length	Second char of buffer = length of input Rest of the buffer = input string followed by carriage return (0Dh)
4Ch (EXIT)	AL=exit code	-

INT: Exit

Exit:

```
MOV AH, 4Ch
```

```
MOV AL, 00      ;Exit code
```

```
INT 21h
```

Subfunction (AH)	Input	Output
4Ch (EXIT)	AL=exit code	-

INT: Print Character

Output_char:

```
MOV AH, 02h
```

```
MOV DL, "!" ; you can put ASCII code too
```

```
INT 21h
```

Subfunction (AH)	Input	Output
02 (DISPLAY_OUPUT) Displays one character on screen	DL=character	-

INT: Print String

Output:

```
MOV DX, OFFSET MSG ; MSG is a variable that holds string
```

```
MOV AH, 09h
```

```
INT 21h
```

Subfunction (AH)	Input	Output
09 (PRINT_STRING) Displays a string on screen String must end with "\$"	DX=address of string	-

INT: Read Input

Read:

```
MOV AH, 01h
```

```
INT 21h           ; Character is in AL
```

Subfunction (AH)	Input	Output
01 (KEYBOARD_INPUT) Reads and displays one character	-	AL=character read

Labels and Jump

- We can have labels that indicates functional parts of the code.
- You can jump to label by JMP command.

First Exercise: Print Char

- In this exercise, we will print `!` in the terminal step by step.

First Exercise: Print Char

- In this exercise, we will print ! in the terminal step by step.

1. Find an appropriate interrupt to print character.

You have to call **INT 21h** with **02 subfunction** (put in AH) and make sure that character in DL register.

Subfunction (AH)	Input	Output
02 (DISPLAY_OUPUT) Displays one character on screen	DL=character	-

First Exercise: Print Char

- In this exercise, we will print ! in the terminal step by step.

1. Find an appropriate interrupt command to print character.

You have to call **INT 21h** with **02 subfunction**(put in AH) and make sure that character in DL register.

Output_char:

```
MOV DL, "!"           ; DL points to ! character
MOV AH, 02             ; subfunction 2 output a character
INT 21h                ; Output the message
```

First Exercise: Print Char

- In this exercise, we will print ! in the terminal step by step.

1. Find an appropriate interrupt command to print character.

You have to call **INT 21h** with **02 subfunction**(put in AH) and make sure that character in DL register.

Output_char:

```
MOV DL, "!"           ; DL points to ! character
MOV AH, 02             ; subfunction 2 output a character
INT 21h                ; Output the message
```

2. Exit

Exit:

```
MOV AH, 4Ch
MOV AL, 00
INT 21h
```

First Exercise: Print Char

- How to skip output char with jump command?
- Put ASCII code into DL instead of character.
- How to print string?

First Exercise: Print Char

- How to skip output char with jump command?
JMP Exit at the beginning.
- Put ASCII code into DL instead of character.
MOV DL, 33
- How to print string?

Second Exercise: Hello World

- In this exercise, we will print hello world in the terminal step by step.

Second Exercise: Hello World

- In this exercise, we will print hello world in the terminal step by step.
1. Define a string (you have to put in the variable)

```
MSG    DB    "Hello World.$"           ; A string variable with a value.
```

Second Exercise: Hello World

- In this exercise, we will print hello world in the terminal step by step.

1. Define a string (you have to put in the variable)

JMP Output

```
MSG    DB    "Hello World.$"           ; A string variable with a value.
```

2. Find an appropriate interrupt command to print string.

You have to call **INT 21h** with **09 subfunction** (put in AH) and make sure that address of the string is in the DX.

Subfunction (AH)	Input	Output
09 (PRINT_STRING) Displays a string on screen String must end with "\$"	DX=address of string	-

Second Exercise: Hello World

- In this exercise, we will print hello world in the terminal step by step.

1. Define a string (you have to put in the variable)

JMP Output

```
MSG    DB    "Hello World.$"           ; A string variable with a value.
```

2. Find an appropriate interrupt command to print string.

You have to call **INT 21h** with **09 subfunction** (put in AH) and make sure that address of the string is in the DX.

Output:

```
MOV DX, OFFSET MSG           ; DX points to (holds the address of) the string
MOV AH, 09                   ; subfunction 9 output a string
INT 21h                      ; Output the message
```

Second Exercise: Hello World

- In this exercise, we will print hello world in the terminal step by step.

1. Define a string(you have to put in the variable)

```
JMP Output
MSG    DB  "Hello World.$"           ; A string variable with a value.
```

2. Find an appropriate interrupt command to print string.

You have to call **INT 21h** with **09 subfunction**(put in AH) and make sure that address of the string is in the DX.

Output:

```
    MOV DX, OFFSET MSG           ; DX points to (holds the address of) the string
    MOV AH,09                   ; subfunction 9 output a string
    INT 21h                     ; Output the message
```

3. Exit

Exit:

```
    MOV AH,4ch
    MOV AL,00                   ; Exit code 0
    INT 21h                     ; Terminate program
```

Second Exercise: Hello World

- How to print new line?

```
cr      dw 13,10,"$"      ; carriage return, line feed
print_cr:
    mov ah,09
    mov dx,offset cr      ; print out a carriage return character
    int 21h
```



```
JMP START
    VAR DB 'Hello, World!$'

START:
    MOV DL, '!'           ; PRINT EXCLAMATION MARK
    MOV AH, 02
    INT 21h

    MOV DL, 10            ; PRINT NEW LINE
    MOV AH, 02
    INT 21h

    MOV AH, 01            ; READ INPUT
    INT 21h              ; AL STORES THE INPUT
    MOV DL, AL            ; TO PRINT THE INPUT
    MOV AH, 02
    INT 21h              ; DO THE PRINTING

    MOV DX, OFFSET VAR    ; PRINT A MESSAGE
    MOV AH, 09
    INT 21h

    MOV AH, 4Ch
    MOV AL, 0
    INT 21h
```



```
code segment
    MOV DL, '!'           ; PRINT EXCLAMATION MARK
    MOV AH, 02
    INT 21h

    MOV DL, 10            ; PRINT NEW LINE
    MOV AH, 02
    INT 21h

    MOV AH, 01            ; READ INPUT
    INT 21h              ; AL STORES THE INPUT
    MOV DL, AL            ; TO PRINT THE INPUT
    MOV AH, 02
    INT 21h              ; DO THE PRINTING

    MOV DX, OFFSET VAR    ; PRINT A MESSAGE
    MOV AH, 09
    INT 21h

    MOV AH, 4Ch
    MOV AL, 0
    INT 21h

    VAR DB 'Hello, World!$'
code ends
```

Arithmetic Operations

INC AL ; Increments value in AL by one and writes in AL again.

DEC AL ; Decrements value in AL by one and writes in AL again.

ADD AX, BX ; Add value in AX and value in BX and writes into AX. ($AX = AX + BX$)

SUB AX, BX ; Value in AX minus value in BX and writes into AX. ($AX = AX - BX$)

Mnemonic	Operands	Size	Operation	Flags
ADD	DEST, SRC	B/W	$(DEST) \leftarrow (DEST) + (SRC)$	AF, CF, OF, PF, SF, ZF
SUB	DEST, SRC	B/W	$(DEST) \leftarrow (DEST) - (SRC)$	AF, CF, OF, PF, SF, ZF

Arithmetic Operations

Multiplication uses AL by default as multiplicand and writes into AX for 8-bit values:

MUL CH ; (AL*CH -> AX)

Mnemonic	Operands	Size	Operation	Flags
MUL	SRC	B	$(AX) \leftarrow (AL) * (SRC)$	CF (carry), OF (overflow)
	SRC	W	$(DX:AX) \leftarrow (AX) * (SRC)$	CF, OF

Arithmetic Operations

Division uses AX by default as dividend and writes result into AL and remainder in AH for 8-bit values:

DIV 7 ;AH = AX % 7,

 ;AL = AX / 7

Mnemonic	Operands	Size	Operation
DIV	SRC	B	(AX) / (SRC) Quotient: AL Remainder: AH
	SRC	W	(DX:AX) / (SRC) Quotient: AX Remainder: DX

Arithmetic Operations

Shift left:

SHL AX, 2 ; Shifts AX left by 2 bits, equivalent to multiply by 4

Shift right

SHR AX, 3 ; Shift AX right by 3 bits, equivalent to divide by 8

Compare: CMP

CMP compares two values and based on the comparison you can jump to different labels. (If else and for loop)

```
CMP AX, 6    ; will compare AX and 6
```

```
CMP AH, BL   ; will compare AH and BL
```

```
var1 DW 0
```

```
CMP AX, var1 ; will compare AX and the value in var1
```

Mnemonic	Operands	Size	Operation	Flags
CMP	DEST, SRC	B/W	(DEST) - (SRC)	AF, CF, OF, PF, SF, ZF

Compare: CMP

Numbers	Condition	Flag Conditions
Signed	(dest) > (src)	SF = OF, ZF = 0
	(dest) < (src)	SF <> OF, ZF = 0
	(dest) = (src)	ZF = 1
Unsigned	(dest) > (src)	CF = 0, ZF = 0
	(dest) < (src)	CF = 1, ZF = 0
	(dest) = (src)	ZF = 1

Conditional Jumps

- JMP - Jump **no matter what** (doesn't check conditions)
- JE - Jump on **equal**
- JNE - Jump on **not equal**
- JNAE - Jump on **not above or equal**
- JA - Jump on **above** (if the first is **greater**)
- JB - Jump on **below** (if the first is **less**)
- JC - Jump on **carry**

Exercise: Loop

```
; A PROGRAM THAT READS AND PRINTS CONSOLE INPUT 5 TIMES
START:
    MOV CL, 00
LOOPING:
    MOV AH, 01      ; READ INPUT FROM CONSOLE
    INT 21h         ; THE INPUT IS STORED IN AL

    MOV DL, AL      ; MOVE INPUT TO OUTPUT REGISTER
    MOV AH, 02      ; PREPARE FOR OUTPUT
    INT 21h         ; OUTPUT
    INC CL           ; INCREMENT THE LOOP COUNTER
    CMP CL, 5        ; CHECK IF 5 ITERATIONS ARE REACHED
    JB LOOPING       ; LOOP
    ;CMP DL, 0Dh     ; CHECK IF INPUT CHAR IS ENTER INSTEAD.
    ;JNE LOOPING     ; LOOP
    MOV AH, 4Ch      ; EXIT PROCEDURE
    MOV AL, 0
    INT 21h
```

Exercise: Print Maximum

Read 16-bit integer inputs from command line, and print out the maximum.

1. Set up variables

```
jmp start

; 5 dup 0 = 0 0 0 0 0. dup 0 means fill array with 0

number db 5 dup 0 ; string which will store output. db because a character is byte long.
5 elements because result will be at most  $2^{15} - 1$  (=32767)

cr      dw 10,"$" ; carriage return, line feed
temp    dw 0

start:
    mov cx,0      ; cx will hold the current integer
    mov bx,0      ; bx will hold the maximum number so far
```

Exercise: Print Maximum

2. Read characters and compare maximum

```
16 morechar:
17     mov ah,01h
18     int 21h    ; reads a character to al
19     mov dx,0    ; dx becomes 0
20     mov dl,al   ; store the input character in dl
21     mov ax,cx
22     cmp dl,0D   ; check if the character is enter (array is over)
23     je compare
24     cmp dl,' '  ; check if the character is space (current integer is finished)
25     je compare
26     sub dx,'0'  ; convert from ascii to real numeric value
27     mov temp,dx ; 3 lines later, when we multiply ax and cx, dx changes. so, copy dx's value to temp
28     mov ax,cx
29     mov cx,10d
30     mul cx      ; multiply ax by 10
31     add ax,temp
32     mov cx,ax   ; put result to cx again
33     jmp morechar
34
35 compare:
36     cmp cx, bx
37     jl compare_end ; if cx < bx then jump to compare_end
38     mov bx, cx     ; bx becomes cx
39
40 compare_end:
41     mov cx, 0      ; now cx is zero, we are ready for the next integer
42     cmp dl,' '     ; only if character was space, go back to reading
43     je morechar
```

17-18 read current character

22 if it is enter compare last integer and end it

24 if it is space compare last integer and wait for the next one

26 convert to numeric value

28-32
last_characters*10+cur_char

For example; if you read 78 up to this point when 3 comes:
 $78*10+3 = 783$

Exercise: Print Maximum

3. Convert decimal to characters

```
45  setup_string:
46      mov ax,bx      ; put result to ax
47      mov bx,offset number+4 ; put a $ at end of buffer
48      mov b[bx],"$"   ; we will fill buffer from back
49      dec bx
50
51  convert_decimal:
52      mov dx,0
53      mov cx,10d
54      div cx          ; divide ax by 10 to get the last digit
55      add dx,48d       ; convert remainder (last digit) to its ASCII representation
56      mov [bx],dl      ; and move to buffer for output
57      dec bx
58      cmp ax,00        ; check if we have got all digits
59      jnz convert_decimal
```

45:

This adds the \$ sign at the end of the address of number and decrements current pointer:
0 0 0 0 0 \$

51:

Divides maximum number(which is in ax) by 10
add char value of it to the current pointer
decrease the pointer
repeats until it sees 0 again

Exercise: Print Maximum

3. Print and close

```
61  print_cr:
62      mov ah,09
63      mov dx,offset cr    ; print out a carriage return character
64      int 21h
65
66  printout:
67      mov dx,bx           ; give the address of string to dx
68      inc dx              ; we decremented once too many, go forward one
69      mov ah,09
70      int 21h
71
72  close:
73      mov ah,04ch         ; exit the program with 0
74      mov al,00
75      int 21h
76
```

61:

Prints new line to avoid overwriting

66:

Prints string of the integer(last value is \$)

72:

Close

Exercise: Find Max Digit

```
; A PROGRAM TO PRINT MAX OF 5 DIGITS ENTERED BY USER
```

```
JMP GREET          ; SKIP DATA PART
```

```
GREETING DB 'ENTER 5 DIGITS $'
```

```
OUTPUTMSG DB 'THE MAX IS: $'
```

```
GREET:
```

```
    MOV DX, OFFSET GREETING ; WELCOME THE USER WITH A MESSAGE
```

```
    MOV AH, 09
```

```
    INT 21h
```

```
    MOV DL, 10          ; PRINT NEW LINE ; MOVE TO NEXT LINE FOR USER INPUT
```

```
    MOV AH, 02
```

```
    INT 21h
```

```
    MOV CL, 00          ; SET COUNTER TO 0.
```

```
    MOV CH, 00          ; MAX WILL BE STORED HERE
```

```
INP:
```

```
    MOV AH, 01          ; READ USER INPUT
```

```
    INT 21h
```

```
    INC CL              ; INCREMENT LOOP COUNTER
```

```
    CMP AL, CH          ; COMPARE THE INPUT WITH EXISTING MAX
```

```
    JA SETCH            ; IF THE USER INPUT IS LARGER, UPDATE MAX
```

```
CHECK:
```

```
    CMP CL, 5           ; CHECK THE LOOP COUNTER
```

```
    JE OUTPUT           ; IF 5 ITERATIONS ARE ACHIEVED, PRINT THE MAX
```

```
    JMP INP             ; ELSE, ASK USER FOR MORE INPUT
```

```
SETECH:
```

```
    MOV CH, AL          ; UPDATE MAX
```

```
    JMP CHECK           ; GO BACK TO ITERATION
```

```
OUTPUT:
```

```
    MOV DL, 10          ; PRINT NEW LINE BEFORE OUTPUT
```

```
    MOV AH, 02
```

```
    INT 21h
```

```
    MOV DX, OFFSET OUTPUTMSG ; PRINT GOODBYE
```

```
    MOV AH, 09
```

```
    INT 21h
```

```
    MOV DL, CH
```

```
    MOV AH, 02
```

```
    INT 21h            ; PRINT MAX
```

```
EXIT:
```

```
    MOV AH, 4Ch ; EXIT PROGRAM
```

```
    MOV AL, 0
```

```
    INT 21h
```

Exercise: Sum

Read 16-bit integer inputs from command line, and print out the sum.

```
jmp start
;=====
; variables
;=====
; 5 dup 0 = 0 0 0 0 0
    number db 5 dup 0      ; stores output string
    cr     dw 10, 13, "$"  ; carriage return, line feed
    temp   dw 0
    n1     dw 0            ; stores first number
    n2     dw 0            ; stores second number
;=====s

start:
    mov cx,0              ; cx will hold the current integer
    mov bl,0              ; bl will be used as counter
```

```
;=====
; read characters
;=====
```

morechar:

```
    mov ah,01h           ; reads a character to al
    int 21h
    mov dx,0             ; dx becomes 0
    mov dl,al            ; store the input character in dl
    mov ax,cx
    cmp dl,0D            ; check if the character is enter (array is over)
    je finnum
    sub dx,'0'           ; convert from ascii to real numeric value
    mov temp,dx          ; 3 lines later, when we multiply ax and cx, dx changes. so, copy dx's value to temp
    mov ax,cx
    mov cx,10D
    mul cx               ; multiply ax by 10 (shift the number read so far) 320 + 5
    add ax,temp          ; add numeric value of new input character to the number read so far
    mov cx,ax            ; put result to cx again
    jmp morechar
```

finnum:

```
    inc bl              ; increment counter
    cmp bl, 1           ; check if first number is over
    je firstnum
    cmp bl, 2           ; check if second number is over
    je secondnum
```

firstnum:

```
    mov n1,cx           ; put first number to n1
    mov cx,0            ; we will release cx to read the next number
    jmp morechar
```

secondnum:

```
    mov n2, cx
```

Exercise: Sum

summation:

```
mov ax, n1
mov bx, n2
add ax, bx
```

```
;=====
; convert number to string
;=====
```

setup_string:

```
;result is already in ax
mov bx,offset number+4 ; put a $ at end of buffer
mov b[bx],"$"          ; we will fill buffer from back
dec bx
```

convert_decimal:

```
mov dx,0
mov cx,10d
div cx ; divide ax (i.e. current number) by 10 to get the last digit
add dx,48d ; convert remainder (last digit) to its ASCII representation
mov [bx],dl ; and move to buffer for output
dec bx
cmp ax,00 ; check if we have got all digits
jnz convert_decimal
```

print_cr:

```
mov ah,09
mov dx,offset cr ; print out a carriage return character
int 21h
```

printout:

```
mov dx,bx ; give the address of string to dx
inc dx ; we decremented once too many, go forward one
mov ah,09
int 21h
```

exit:

```
mov ah,04ch ; exit the program with 0
mov al,00
```

Exercise: Store & Print



```
; A PROGRAM TO STORE AND PRINT 5 INPUTS FROM USER
JMP GREET
INPUT DB 5 dup 0
GREETING DB 'ENTER 5 DIGITS $'
OUTPUTMSG DB 'THE INPUT IS: $'
GREET:
    MOV DX, OFFSET GREETING ; GREET USER
    MOV AH, 09
    INT 21h

    MOV DL, 10                ; PRINT NEW LINE BEFORE READING INPUT
    MOV AH, 02
    INT 21h

    MOV CL, 00                ; SET LOOP COUNTER TO 0.
;    MOV BX, OFFSET INPUT
    MOV BX, 50h                ; SET AN ADDRESS TO USE AS ARRAY HEAD
```

```
INP:
    MOV AH, 01                ; READ INPUT
    INT 21h
    INC CL

    MOV [BX], AL                ; STORE INPUT TO THE POINTED ADDRESS
    ADD BX, 1                    ; INCREMENT THE ADDRESS

CHECK:
    CMP CL, 5                    ; CHECK LOOP END
    JE OUTPUT
    JMP INP

OUTPUT:
    MOV DL, 10                ; PRINT NEW LINE FOR OUTPUT
    MOV AH, 02
    INT 21h

    MOV DX, OFFSET OUTPUTMSG ; PRINT GOODBYE
    MOV AH, 09
    INT 21h
```

Exercise: Store & Print

```
GOBACK:
    SUB BX, 1                ; MOVE ARRAY POINTER TO THE BEGINNING
    DEC CL
    CMP CL, 0
    JA GOBACK

OL:
    MOV DL, [BX]             ; READ INPUT FROM MEMORY
    MOV AH, 02              ; PRINT THE ELEMENT
    INT 21h                 ; DO THE PRINTING

    ADD BX, 1               ; MOVE TO NEXT ELEMENT
    INC CL                  ; LOOP UNTIL 5
    CMP CL, 5
    JB OL

EXIT:
    MOV AH, 4Ch             ; EXIT
    MOV AL, 0
    INT 21H
```

Exercise: Reverse String



```
jmp readloop
readloop:
```

```
    MOV AH, 01h        ; to read character
    INT 21h
    CMP AL, 0dh        ; check if it is enter
    JE stopreading
    MOV AH, 0          ; make the top half zero
    push AX            ; push AX to stack
    jmp readloop
```

```
stopreading:
```

```
    ; we will print new line here, different approach than previous one
    MOV AH, 02h        ; print a character
    MOV DL, 0Dh        ; move the cursor to start of the current row, meaning \r
    INT 21h
    MOV DL, 0Ah        ; move the cursor to the next line without changing the column, meaning \n.
    ; Also we don't have to put 02h in AH again because it already is.
    INT 21h
```


Exercise: Reverse String

```
printloop:
    CMP SP, 0FFFEh    ; check if the stack is empty or not
    JE exit          ; if equal jump to exit
    MOV AH, 02h       ; print a character
    POP DX           ; put a character in DL by popping value from stack in DX because top half is empty
    INT 21h
    JMP printloop
exit:
    MOV AH, 04Ch
    MOV AL, 00
    INT 21h
```


References

1. <http://www.csn.ul.ie/~darkstar/assembler>
2. <https://fruttenboel.verhoeven272.nl/asm/a86man.htm>
3. https://www.wikiwand.com/en/Assembly_language
4. https://www.wikiwand.com/en/Machine_code
5. <https://cysecguide.blogspot.com/2016/12/ascii-code-table.html>
6. <https://patater.com/gbaguy/x86asm.htm>